Professor Fearing EECS192/Assignment #1 v1.00 Spring 2017

Due at beginning of class Tues Mar. 14.

(One assignment per team.)

The purpose of this assignment is to allow you to prototype track finding algorithms with a high level language before implementing in C on the KL25Z processor. You are free to use signal processing libraries, floating point, etc. for initial prototyping. However, remember that your line finding algorithm will need to work in real-time on a 40 MIPS processor., so your final Python should not be dependent on libraries which won't run in real time on mbed.

Typical strategies to use for finding the track include:

1. Frame subtraction and peak detection
2. smoothing followed by gradient detection (e.g. difference of Gaussians approximation to the Laplacian)
3. curve fitting, e.g. cubic spline or $\tanh^{-1}$.

A set of line scan data from EECS192 2016 Team 1: Fast and Curious is provided on Piazza for EE192 under "Resources", `natcar2016_team1.csv`. A Python template is provided `linescanplotsHW1.py` which will read the csv file (can easily be modified to read telemetry file by adding extra columns) and plot line camera data and velocity. The actual car run can be seen at `https://www.youtube.com/watch?v=_AMs9iixp5M`. A bit more then a complete lap is given in data, as you can see the steps right after the start line. This data has been pre-processed to control illumination. Note that the time sampling is not uniform as darker areas use longer exposure (this is not necessarily the best strategy). Also, the velocity data is quite noisy.

Complete the function `find_track(linescans)` which takes as input n frames of linescans of 128 values in the range 0...255 and returns:

a) `track_center_list` which is a length n list of the index in the range 0...127 corresponding to the center of the track in each frame.

b) `track_found_list` which is a length n list of booleans, **True** if the track is visible for a particular frame.

c) `Cross_found_list` which is a length n list of booleans, **True** if a crossing is present for a particular frame.

Upload your completed Python code to bcourses, as well as .png plots of track center as function of time, and plots of track found and crossing found.

Your python function will be tested against another data set taken under similar conditions on a similar track.

The line scan data should be interpreted as given in the following table. For example, around scan 380, the track is out of frame, so `track_found` would be False.

Input track?	Input crossing?	output track found	output cross found
F	F	F	F
F	T	F	T
T	F	T	F
T	T	T	T